REMARKS/ARGUMENTS

Claims 1, 2, 4-12, and 14-20 are pending in the present application. Claims 1, 7, 9, 11, 12, and 14-20 are amended. Support for amendments to the claims may be found in the specification on page 4 lines 5-30, page 11 lines 11-13, page 13 line 21, page 14 line 2, page 14 lines 8-17, page 21 lines 8-24, page 22 line 25 and page 23 line 2. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 101

The Examiner has rejected claims 19-20 under 35 U.S.C. § 101 as being directed towards non-statutory subject matter. The Examiner states:

> Claim 19 recites an apparatus comprising means for performing function steps. It is clear from the specification and claim 11, that those steps in at least one embodiment are performed by software. This shows that there is at least one embodiment of the claim that is directed towards software only, which is non-statutory subject matter.

Office Action dated August 20, 2008, page 2.

Claim 19 is as follows:

> An apparatus for generating a logically merged web module for a web application, comprising:
>
> a system bus;
>
> a local memory connected to the system bus, wherein the memory contains computer executable instructions;
>
> a processor connected to the system bus, wherein the processor executes the computer executable instructions to direct the apparatus to:
>
> responsive to a determination that a shared module designation file exists, identify at least one shared web module from the shared module designation file to be incorporated into a web application, to form at least one identified shared web module, wherein the shared web module designation file includes all descriptors that reference the at least one shared web module;
>
> locate the at least one identified shared web module; and
>
> logically merge the at least one shared web module with web modules of the web application, in accordance with the shared web module designation file to generate a logically merged web application, wherein a reference to the at least one shared web module is used rather than a copy of the at least one shared web module in the logically merged web application.

Applicants have amended claim 19 to incorporate statutory subject matter of the machinery of the claim. The claim is directed to a combination of hardware and software which is statutory. Therefore, the amendment provided overcomes the rejection of claim 19. Further, the statutory subject matter of claim 19 forms a base upon which claim 20 depends. Therefore, claim 20 also overcomes the rejection. Accordingly, amended claims 19 and 20 overcome the stated rejection under 35 U.S.C. § 101.

II.A. 35 U.S.C. § 103, Obviousness

The Examiner has rejected claims 1-2, 4, 6-12 and 14-20 under 35 U.S.C. § 103 as being obvious over Li, Computer-Implemented Sharing of Java Classes for Increased memory Efficiency and Communication Method, U.S. Patent 6,519,594, (February 11, 2003), (hereinafter "*Li*"), in view of *Spotswood*, System and Method for Using a Class Loader Hierarchy to Load Software Applications, U.S. Application Publication 2004/0255293, (December 16, 2004), (hereinafter "*Spotswood*"). The Examiner states:

> **Regarding claims 1, 11, and 19**, Li teaches a method of generating a logically merged web module for a web application, comprising:
>
> determining if the application includes a reference to at least one shared module, that is capable of being incorporated into a plurality of applications (Col. 8, lines 10 -21), in a shared module designation file, wherein the shared description file includes all descriptors that reference the shared module (Col. 9, lines 16 - 32);
>
> specifying a path to a location of the at least one shared module (Col. 9, lines 16-32); and
>
> logically merging the at least one shared module with modules of the web application, in accordance with the shared module designation file to generate a logically merged web application (Col. 10, lines 26 - 49).
>
> Li does not explicitly indicate that the applications and modules comprise web applications and its modules.
>
> *Spotswood* teaches that web applications can be run using java virtual machine on a web server (¶56-62).
>
> It would have been obvious to one of ordinary skill in the art at the time the invention was made to use *Spotswood*'s teaching of web applications and their use of web modules and java classes to expand Li's teaching of shared java classes to include web applications running in java virtual machine.

Office Action dated August 20, 2008, pp. 3-4.

The Examiner bears the burden of establishing a *prima facie* case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). The prior art reference (or references when combined) must teach or suggest all the claim limitations. *In re Royka,* 490 F.2d 981, 180 USPQ 580 (CCPA 1974). In determining obviousness, the scope and content of the prior art are… determined; differences between the prior art and the claims at issue are… ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background, the obviousness or non-obviousness of the subject matter is determined. *Graham v. John Deere Co.*, 383 U.S. 1 (1966). "Often, it will be necessary for a court to look to interrelated teachings of multiple patents; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by a person having ordinary skill in the art, all in order to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue." *KSR Int'l. Co. v. Teleflex, Inc.*, 127 S. Ct. 1727 (April 30, 2007). "*Rejections on*

obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the *legal conclusion* of obviousness. Id. (citing *In re Kahn*, 441 F.3d 977, 988 (CA Fed. 2006))."

Claim 1 is as follows:

> A method of generating a logically merged web module for a web application, comprising:
>
> responsive to a determination that a shared module designation file exists, identifying at least one shared web module from the shared module designation file to be incorporated into a web application, to form at least one identified shared web module, wherein the shared web module designation file includes all descriptors that reference the at least one shared web module;
>
> locating the at least one identified shared web module using path information;
>
> logically merging the at least one shared web module with web modules of the web application, in accordance with the shared web module designation file to generate a logically merged web application, wherein a reference to the at least one shared web module is used in the logically merged web application rather than a copy of the at least one shared web module.

Li, in combination with *Spotswood,* does not teach each and every feature of claim 1 as they are arranged in the claim. Therefore, *Li* and *Spotswood* fail to provide a *prima facie* obviousness rejection for the claimed features. In particular, *Li* and *Spotswood* fail to teach the features of "responsive to a determination that a shared module designation file exists, identifying at least one shared web module from the shared module designation file to be incorporated into a web application, to form at least one identified shared web module, wherein the shared web module designation file includes all descriptors that reference the at least one shared web module", "locating the at least one identified shared web module using path information," and "logically merging the at least one shared web module with web modules of the web application, in accordance with the shared web module designation file to generate a logically merged web application, wherein a reference to the at least one shared web module is used in the logically merged web application rather than a copy of the at least one shared web module." The Examiner has asserted portions of *Li* and *Spotswood* in support of the rejection to be addressed by the Applicants.

With regard to the feature of "responsive to a determination that a shared module designation file exists, identifying at least one shared web module from the shared module designation file to be incorporated into a web application, to form at least one identified shared web module, wherein the shared web module designation file includes all descriptors that reference the at least one shared web module," the following portion of *Li* is asserted:

> With reference to FIG. 7, the shared memory pool 280 is shown interacting with a JavaLayer Class Manager (JCM) 270 and a shared lock 260. When a JVM initializes its class library, the JCM 270 uses the shared lock 260 to serialize access to the shared memory pool 280, also called the class pool 280. The shared memory pool 280 is an area of memory that is established for storing Java classes and other information to be shared across multiple JVMs which are running simultaneously on system 112. In this way, the shared lock 260 is used by the present invention for serializing access of shared Java classes between multiple Java applications 256-258.

Li, col. 8 lines 10-21.

> FIG. 8 illustrates physical and logical memory organization of the classes within the memory pool 280. To avoid the physical copying and moving of classes when operations are performed on the classes by multiple JVMs 251-253, these classes are stored in memory 310 in the manner as shown in FIG. 8. The class is stored in variable sized class cells or "blocks" 290-294. The actual class is constructed by linking cells together at certain offsets in the class cells as shown by 324a-324c. The size of any block can be dynamically adjusted based on the size of the stored Java class. The link contains the name of the class, its size and the offset from the start. The entries in the name table are shown as 322a-322c. A hash table 320 is used to reference the position within the name table for an enumerated Java class. The name is then used to reference an offset which points to the memory position of the class within the shared memory pool 280.

Li, col. 9, lines 16-32.

Li is directed toward:

> ... allowing Java classes to be shared among many Java virtual machines (JVMs) including a communication system allowing Java and native applications to readily interoperate. ... The present invention provides a shared memory pool (SMP) into which a JVM and store and register a particular Java class. The stored and registered Java class is then accessible by other JVMs using the SMP and a Java layer class manager that is implemented in software.

Li, Abstract.

Li, therefore teaches a memory sharing mechanism for Java virtual machines. Further, *Li* teaches in the cited portion, "the shared lock 260 is used by the present invention for serializing access of shared Java classes between multiple Java applications 256-258." *Li* also teaches use of a shared lock to serialize use of the shared Java class.

In contrast, the claimed feature responds to a determination that a shared module designation file exists. *Li* does not teach determining the existence of a shared module designation file associated with an application. *Li* teaches a shared class pool for storing shared Java classes, without a determination of which class may belong to an application. *Li* further requires preloading of the shared classes into the memory pool. Further, *Li* fails to teach the web module designation file includes all descriptors that

reference the at least one shared web module because *Li* does not teach a web module designation file. Therefore, *Li* does not teach the claimed feature.

With regard to the feature of, "locating the at least one identified shared web module using path information," *Li* teaches use of a memory address of a class as in:

> FIG. 8 illustrates physical and logical memory organization of the classes within the memory pool 280. To avoid the physical copying and moving of classes when operations are performed on the classes by multiple JVMs 251-253, these classes are stored in memory 310 in the manner as shown in FIG. 8. The class is stored in variable sized class cells or "blocks" 290-294. The actual class is constructed by linking cells together at certain offsets in the class cells as shown by 324a-324c. The size of any block can be dynamically adjusted based on the size of the stored Java class. The link contains the name of the class, its size and the offset from the start. The entries in the name table are shown as 322a-322c. A hash table 320 is used to reference the position within the name table for an enumerated Java class. The name is then used to reference an offset which points to the memory position of the class within the shared memory pool 280.

Li, col. 9 lines 16-32.

Further *Li* teaches:

> For performance reasons physical copying and moving a class is avoided as much as possible during operations. Therefore, all classes 290-296 are managed by the central JCM 270 meta-object which manages the memory pool 280 of class cells that hold all class memory. This memory pool 280 is shared between the multiple JVMs 251-253 and all Java Layer modules as shown in FIG. 7. The JCM 270 allocates memory to the classes within the shared memory pool 280 and divides the shared memory pool 280 into blocks for this purpose. The individual blocks are then managed by the JCM 270. If a JVM wants to share a class, it first checks with the JCM 270 which then references a name table and the lock for the class. The *JCM 270 then forwards the address of the class* to the requesting JVM when the lock is free. The JVM can then access the shared Java class.

Li, col. 8 lines 40-55. (emphasis provided)

In contrast, the claimed feature uses path information to locate a shared module. The shared module is not required to be in memory as in the teaching of *Li*. Therefore, *Li* does not teach the feature as claimed.

With regard to the feature of "logically merging the at least one shared web module with web modules of the web application, in accordance with the shared web module designation file to generate a logically merged web application, wherein a reference to the at least one shared web module is used in the logically merged web application rather than a copy of the at least one shared web module," *Li* in the previously cited portion teaches: "...these classes are stored in memory 310 in the manner as shown in

FIG. 8. The class is stored in variable sized class cells or "blocks" 290-294..." *Li* therefore teaches loading of a class into a memory location for use by other Java virtual machines. A copy of the class is loaded physically into the memory pool.

Further, *Li* teaches access to a shared memory pool resource of a Java class as in:

> The remainder of the steps of FIG. 10 illustrate the procedure used for accessing the stored class(N) that resides in the shared memory pool. At step 418, a JVM, e.g., JVM(J), checks with the JCM to determine whether or not a class(N) is stored in the shared memory pool. It is appreciated that JVM(I) and JVM(J) are both running simultaneously on system 112 (or could be the same JVM). The above determination is typically done by an instruction whose function is to establish class(N) for the JVM. During the establishment function, before the class is written into memory, a check is first done to determine if the class has already been established in the shared memory pool by another JVM. At step 420, if class(N) exists in the shared memory pool, then JVM(J) requests and obtains the read key for class(N) from the JCM. At step 422, JVM(J) then accesses class(N), as needed, from the shared memory pool instead of copying its own instantiation of class(N) in memory. At step 424, JVM(J) then releases the read key for class(N) when done with this class. It is appreciated that steps 418-424 can be repeated for other JVMs that are simultaneously running on computer system 112. Memory is thereby saved because each JVM that needs class(N) can share the instantiation of this class as established by JVM(I) without copying its own version.

Li, col. 10 lines 26-50.

In contrast the claimed feature uses, "a reference to the at least one shared web module is used in the logically merged web application rather than a copy of the at least one shared web module." The claimed feature does not load a class into memory or obtain a read key for the memory as taught by *Li*. Rather, the claimed feature includes a reference for the shared module into the logically merged web application. Therefore, *Li* fails to teach the claimed feature.

The Examiner notes that *Li* does not explicitly indicate the applications and modules comprise web applications and modules. However, *Spotswood* teaches the missing feature of web applications. *Spotswood* is asserted to teach:

> [0056] WebLogic Server automatically creates a hierarchy of classloaders when an application is deployed. The root classloader in this hierarchy loads any EJB JAR files in the application. A child classloader is created for each Web Application WAR file. Because it is common for Web Applications to call EJBs, the application classloader architecture allows JavaServer Page (JSP) files and servlets to see the EJB interfaces in their parent classloader. This architecture also allows Web Applications to be redeployed without redeploying the EJB tier. In practice, it is more common to change JSP files and servlets than to change the EJB tier.

> [0057] FIG. 3 illustrates this WebLogic Server application classloading concept. If the application includes servlets and JSPs that use EJBs, the developer should:
>
> [0058] Package the servlets and JSPs in a WAR file.
>
> [0059] Package the enterprise beans in an EJB JAR file.
>
> [0060] Package the WAR and JAR files in an EAR file.
>
> [0061] Deploy the EAR file.
>
> [0062] Although the developer could deploy the WAR and JAR files separately, deploying them together in an EAR file 150 produces a classloader arrangement that allows the servlets and JSPs to find the EJB classes. If they deploy the WAR and JAR files separately, the server 144 (in this instance the WebLogic Server) creates sibling classloaders 146, 148 for them. This means that they must include the EJB home and remote interfaces in the WAR file, and the server must use the RMI stub and skeleton classes for EJB calls, just as it does when EJB clients and implementation classes are in different JVMs. The Web application classloader 152, 154 contains all classes for the Web application except for the JSP class. The JSP class 156, 158, 160 obtains its own classloader, which is a child of the Web application classloader. This allows JSPs to be individually reloaded.

Spotswood paragraphs [0056] – [0062].

Spotswood appears to teach use of a Web application class loader. However, *Spotswood* fails to teach the claimed features previously shown to be missing from the teaching of *Li*. The addition of the teaching of *Spotswood* therefore does not overcome the missing features from *Li*. The combination of the teaching of *Li* and *Spotswood* does not teach each and every feature of claim 1 as the features are arranged in the claim.

Accordingly, the combination of *Li* and *Spotswood*, when considered as a whole, fail to teach the features of claim 1. Independent claims 11 and 19 have similar distinguishing features as claim 1 and are therefore also distinguished from the teaching of *Li* and *Spotswood*. Dependent claims 2, 4, 6-10, 12, 14-18 and 20 depend from claims 1, 11 and 19 respectively, thereby inheriting the distinguishing features and are therefore also distinguished from the combined teaching of *Li* and *Spotswood*. Therefore, the rejection of claims 1-2, 4, 6-12 and 14-20 under 35 U.S.C. § 103 has been overcome.

With regard to claims 2, 12 and 20, the Examiner further states:

> Regarding claims 2, 12, and 20, Li teaches the method of claims 1, 11, and 19, further comprising: loading the logically merged application into a container (Col. 9, lines 5 - 15).
> Li does not explicitly indicate a web application running on a web container. *Spotswood* teaches that web applications can be run using java virtual machine on a web server (756-62), where the applications are run using on a web container (776).
> It would have been obvious to one of ordinary skill in the art at the time the invention was made to use *Spotswood*'s teaching of web applications and their use of web modules and java classes to expand Li's teaching of shared java classes to include web applications running in java virtual machine.

Office action of August 20, 2008 page 4.

Claim 2 is as follows:

> The method of claim 1, further comprising:
> loading the logically merged web application into a web container.

With regard to the feature of "loading the logically merged web application into a web container," the Examiner believes *Li* and *Spotswood* teach the feature. The Examiner asserts the following portion of *Li*:

> The JavaLayer framework offers APIs to the JCM 270 for JVMs to perform operations on various classes and access the memory contents of the classes. The JavaLayer assumes a basic container, called class cells, that is used to support class loading. Class cells are dynamic memory buffers, shared by JVMs in a shared mode, and used for class manipulations avoiding physical copying of class as much as possible. When physical drivers and application objects use the same class cells, they can share the class cell and effectively result in a JVM architecture that does not contain copies.

Li, col. 9 lines 5-15.

Li teaches loading classes into class cells that are memory buffers shared by JVMs. The class cells contain only the shared classes. In contrast, the claimed feature loads a logically merged application into a container. Class cells as taught by *Li* therefore do not contain the logically merged application. The class cells of *Li* only contain shared classes and not a logically merged application. Therefore, *Li* does not teach the claimed feature.

Although the Examiner notes *Li* does not teach a web application running in a web container, the Examiner asserts *Spotswood*, as previously stated, for the teaching of "web applications can be run using a Java virtual machine on a web server (paragraphs 56-62), where the applications are run using on a web container (paragraph 76)." As previously shown, *Li* and *Spotswood* do not teach the features of claim 1

upon which claim 5 is based. Neither *Li* nor *Spotswood* teach logically merging an application, therefore neither reference teaches loading a logically merged application as claimed.

Therefore, the teaching of *Li* in combination with *Spotswood* when viewed as a whole does not teach the claimed feature of claim 5. Therefore, the rejection of claims 2, 12 and 20 under 35 U.S.C. § 103 has been overcome.

Regarding claim 4, the Examiner asserts the following:

> Regarding claim 4, Li teaches the method of claim 1.
> Li does not explicitly indicate wherein the web application is an enterprise archive (EAR) and wherein the logically merged web application is a logically merged EAR.
> *Spotswood* teaches that web applications can be run using java virtual machine on a web server (756-62), indicate wherein the web application is an enterprise archive (EAR) and wherein the logically merged web application is a logically merged EAR.
> It would have been obvious to one of ordinary skill in the art at the time the invention was made to use *Spotswood*'s teaching of web applications and their use of web modules and java classes to expand Li's teaching of shared java classes to include web applications running in java virtual machine.

Office action of August 20, 2008 pp. 4-5.

Claim 4 is as follows:

> The method of claim 1, wherein the web application is an enterprise archive (EAR) and wherein the logically merged web application is a logically merged EAR.

With regard to the feature of "the web application is an enterprise archive (EAR) and wherein the logically merged web application is a logically merged EAR," as previously shown, *Li* and *Spotswood* do not teach the claimed features of claim 1. Therefore, the base upon which claim 4 depends is neither taught nor suggested by *Li* and *Spotswood*. The Examiner asserts paragraphs [0056]-[0062], as previously stated, and the following portion of *Spotswood* in teaching use of an EAR file:

> [0049] In a typical application server environment, for example a WebLogic Server implementation, classloading is centered on the concept of an application. An application is normally packaged in an Enterprise Archive (EAR) file containing application classes. Everything within an EAR file is considered part of the same application. The following may be part of an EAR or can be loaded as standalone applications:
>
> [0050] An Enterprise JavaBean (EJB) JAR file;
>
> [0051] A Web Application WAR file; and/or,

> [0052] Resource Adapter RAR file.
>
> [0053] If a developer deploys an EJB JAR file and a Web Application WAR file separately, they are considered two applications. If they are deployed together within an EAR file, they are considered a single (i.e. one) application. The developer can deploy components together in an EAR file for them to be considered part of the same application. If the developer needs to use resource adapter-specific classes with Web components (for example, an EJB or Web application), they must bundle these classes in the corresponding component's archive file (for example, the JAR file for EJBs or the WAR file for Web applications).

Spotswood, paragraphs 0049-0053.

Spotswood teaches the EAR file is an optional method of deploying a package of files containing content of a WAR file and a EJB JAR file. The file contents are the physical files. In contrast, the logically merged application contains a reference to a shared module and not the shared module itself. Therefore, neither *Li* nor *Spotswood* teaches a logical merged application as claimed. Therefore, *Li* in combination with *Spotswood* do not teach the claimed feature of claim 4. Therefore, the rejection of claim 4 under 35 U.S.C. § 103 has been overcome.

With regard to claims 6 and 14, the Examiner states the following:

> Regarding claims 6 and 14, Li teaches the method of claims 1 and 11, wherein logically merging the at least one shared web module with web modules of the web application includes: determining a priority associated with the at least one shared web module (Col. 9, lines 59 - 67); and resolving any conflicts between shared web modules in the at least one shared web module and conflicts between the at least one shared web module and web modules of the web application, if any (Col. 8, lines 10 - 21).

Office action of August 20, 2008 page 5.

Claim 6 is as follows:

> The method of claim 1, wherein logically merging the at least one shared web module with web modules of the web application includes:
>
> determining a priority associated with the at least one shared web module; and
>
> resolving any conflicts between shared web modules in the at least one shared web module and conflicts between the at least one shared web module and web modules of the web application.

With regard to the claimed feature of "determining a priority associated with the at least one shared web module," as previously shown, *Li* does not teach the features of claims 1 and 11. The Examiner asserts the following portion of *Li*:

> As shown in FIG. 9, the JCM 270 locks the class cells which are static classes or static variables to restrict the write-right. It gives the write-right to the object which is writing the static classes. JavaLayer allocates the class cell to the static classes with one write-lock (WL), which is used to record the object name. The highest priority is given to the writing object to prevent dead lock.

Li, col. 9, lines 59-67.

Li teaches assignment of lock to the class cells to restrict the write right. Lock management gives a highest priority to a writing object to prevent deadlock. In contrast, the claimed feature determines a priority associated with at least one shared web module when performing a logical merge of the shared web modules with web modules of the application. The teaching of *Li* is directed to a lock management mechanism and has nothing to do with logically merging application modules. The claimed feature does not deal with locking at all.

With regard to the feature of "resolving any conflicts between shared web modules in the at least one shared web module and conflicts between the at least one shared web module and web modules of the web application," *Li* teaches the use of locks as in the following:

> With reference to FIG. 7, the shared memory pool 280 is shown interacting with a JavaLayer Class Manager (JCM) 270 and a shared lock 260. When a JVM initializes its class library, the JCM 270 uses the shared lock 260 to serialize access to the shared memory pool 280, also called the class pool 280. The shared memory pool 280 is an area of memory that is established for storing Java classes and other information to be shared across multiple JVMs which are running simultaneously on system 112. In this way, the shared lock 260 is used by the present invention for serializing access of shared Java classes between multiple Java applications 256-258.

Li, col. 8 lines 10-21.

The use of locks is not required in the claimed feature. Locks, as taught by *Li,* are used after the shared class is in the memory pool and not before the application is logically merged. Therefore, *Li* teaches away from the claimed feature by using locks and a lock process occurring after the shared class has been loaded.

Therefore, the teaching of *Li* does not teach the claimed feature of claim 6. Because claim 14 has similar subject matter as claim 6, *Li* fails to teach the feature of claim 14. Therefore, the rejection of claims 6 and 14 under 35 U.S.C. § 103 has been overcome.

With regard to claims 7 and 15, the Examiner states the following:

> Regarding claims 7 and 15, Li teaches the method of claims 1 and 11, wherein the steps of determining, specifying, and logically merging are performed during

> an initialization process of a runtime environment for initializing the web application to be run on a server (Col. 8, lines 40 - 62).

Office action of August 20, 2008 page 5.

Claim 7 is as follows:

> The method of claim 1, wherein the steps of identifying, locating, and logically merging are performed during an initialization process of a runtime environment for initializing the web application to be run on a server.

With regard to the feature of "wherein the steps of identifying, locating, and logically merging are performed during an initialization process of a runtime environment for initializing the web application to be run on a server," *Li* is asserted to teach the claimed feature in the following:

> For performance reasons physical copying and moving a class is avoided as much as possible during operations. Therefore, all classes 290-296 are managed by the central JCM 270 meta-object which manages the memory pool 280 of class cells that hold all class memory. This memory pool 280 is shared between the multiple JVMs 251-253 and all JavaLayer modules as shown in FIG. 7. The JCM 270 allocates memory to the classes within the shared memory pool 280 and divides the shared memory pool 280 into blocks for this purpose. The individual blocks are then managed by the JCM 270. If a JVM wants to share a class, it first checks with the JCM 270 which then references a name table and the lock for the class. The JCM 270 then forwards the address of the class to the requesting JVM when the lock is free. The JVM can then access the shared Java class.
> As shown in FIG. 7, an application can be associated with each JVM. As shown, application 256 is associated with JVM 251, application 257 is associated with JVM 252 and application 258 is associated with JVM 253. Requests from the JVM to access Java classes 290-296 are processed by the JCM 270. A shared lock 260 used so that only one JVM can access any particular Java class at any time.

Li, col. 8 lines 40-62.

Li teaches allocating memory for a shared memory pool further divided into blocks into which shared classes are stored. In contrast, the claimed feature is directed to steps of identifying, locating and logically merging an application during an initialization process of a runtime environment for initializing the web application to be run on a server. Memory allocation, as taught by *Li,* is therefore not equivalent to the events of the claimed feature. The initialization taught by *Li* does not encompass the steps of the claimed feature. Therefore, *Li* fails to teach the claimed feature of claims 7 and 15. Therefore, the rejection of claims 7 and 15 under 35 U.S.C. § 103 has been overcome.

.

With regard to claim 8, the Examiner states the following:

> Regarding claim 8, Li teaches the method of claim 1, wherein logically merging the at least one shared web module with the web modules of the web application includes using a service provider interface (SPI) that provides merge logic for merging different module types (Col. 8, lines 40 - 62).

Office action of August 20, 2008 page 5.

Claim 8 is as follows:

> The method of claim 1, wherein logically merging the at least one shared web module with the web modules of the web application includes using a service provider interface (SPI) that provides merge logic for merging different module types.

With regard to the claimed feature of "wherein logically merging the at least one shared web module with the web modules of the web application includes using a service provider interface (SPI) that provides merge logic for merging different module types," *Li* is asserted to teach the same feature at the previously cited reference. *Li* is directed to memory management through allocation of cells in blocks and to lock management. In contrast, the claimed feature merges shared web modules with web modules of web applications using a service provider interface (SPI) that provides merge logic for merging different module types. *Li* fails to teach different shared module types and providing merge logic to help accomplish the merging process. Therefore, *Li* fails to teach the feature of claim 8. Therefore, the rejection of claim 8 under 35 U.S.C. § 103 has been overcome.

With regard to claims 9 and 17, the Examiner states the following:

> Regarding claims 9 and 17, Li teaches the method of claims 2 and 12, wherein the container uses one or more application program interfaces (APIs) to identify the path to the at least one shared web module and loads the at least one shared web module when loading the logically merged web application (Col. 9, lines 5 - 9).

Office action of August 20, 2008 page 5.

Claim 9 is as follows:

> The method of claim 2, wherein the container uses one or more application program interfaces (APIs) to identify a path to the at least one shared web module and loads the at least one shared web module when loading the logically merged web application.

With regard to the claimed feature of "wherein the container uses one or more application program interfaces (APIs) to identify a path to the at least one shared web module and loads the at least one shared web module when loading the logically merged web application," *Li* is asserted to teach the claimed feature in:

> The JavaLayer framework offers APIs to the JCM 270 for JVMs to perform operations on various classes and access the memory contents of the classes. The JavaLayer assumes a basic container, called class cells, that is used to support class loading. Class cells are dynamic memory buffers, shared by JVMs in a shared mode,

Li, col. 9 lines 5-9.

Li teaches providing APIs to the JCM 270 for the JVMs to perform various operations on the classes and to access the memory. *Li* does not teach use of a container as in the claimed feature. Further, *Li* does not teach identifying a path to the at least one shared web module. APIs provided by *Li* are for accessing already loaded cells. Further, *Li* teaches physical cell access is by address, as in "JCM 270 then forwards the address of the class to the requesting JVM when the lock is free," of the previously cited section of col. 8 lines 40-62. In contrast, the claimed feature uses an identified path. Therefore, *Li* fails to teach the feature of claim 9 as claimed. Claim 17 claims similar features and is therefore also distinguished from the teaching of *Li*. Therefore, the rejection of claims 9 and 17 under 35 U.S.C. § 103 has been overcome.

With regard to claims 10 and 18, the Examiner states the following:

> Regarding claims 10 and 18, Li teaches the method of claims 1 and 11, wherein logically merging the at least one shared web module with web modules of the web application includes at least one of re-linking references to the at least one shared web module in the web modules of the web application, extrapolating policy information for the at least one shared web module from a policy file associated with the web application, and modifying a class path for the web application to include paths to each of the at least one shared web modules (Col. 9, lines 16 - 31; lines 61 - 67).

Office action of August 20, 2008 page 6.

Claim 10 is as follows:

> The method of claim 1, wherein logically merging the at least one shared web module with web modules of the web application includes at least one of relinking references to the at least one shared web module in the web modules of the web application, extrapolating policy information for the at least one shared web module from a policy file associated with the web application, and

> modifying a class path for the web application to include paths to each of the at least one shared web modules.

As previously shown, *Li* and *Spotswood* fail to teach the features of claims 1 and 11. With regard to the feature of "logically merging the at least one shared web module with web modules of the web application includes at least one of relinking references to the at least one shared web module in the web modules of the web application," *Li,* as previously shown, fails to teach logically merging. Further, *Li* fails to teach relinking as claimed. The linking in *Li* is with regard to changes in the blocks and not associating changed or additional blocks with an application. In the cited reference, *Li* teaches a linking of cells together and not linking a shared web module with an application as claimed. Therefore, *Li* fails to teach the feature as claimed.

With regard to the feature of "extrapolating policy information for the at least one shared web module from a policy file associated with the web application, and modifying a class path for the web application to include paths to each of the at least one shared web module," *Li* teaches nothing. *Li* does not teach any reference to policy information contained in a policy file associated with the web application. Therefore, *Li* does not teach the feature as claimed.

Because *Li* fails to teach both claimed features, *Li* fails to teach the features of claim 10. Claim 18 has similar features and is therefore also not taught by *Li*. Accordingly, *Li* fails to teach the features of claims 10 and 18. Therefore, the rejection of claims 10 and 18 under 35 U.S.C. § 103 has been overcome.

II.B. 35 U.S.C. § 103, Obviousness

The Examiner has rejected claim 5 under 35 U.S.C. § 103 as being obvious over *Li* in view of *Spotswood*, and further in view of *Sharma*, Modular and Portable Deployment of a Resource Adapter in an Application Server, U.S. Patent 6,721,777, (April 13, 2004), (hereinafter "*Sharma*"). The Examiner states:

> **Regarding claim 5**, the combined teaching of Li and *Spotswood* disclose method of claim 1.
>
> The combination does not explicitly indicate wherein the at least one shared web module includes at least one of a web archive (WAR) file, an enterprise java bean (EJB) archive file, and a resource archive (RAR) file.
>
> *Sharma* teaches that in addition to the java classes that are shared between applications taught in Li, that web applications can also share resource adaptor modules (Col. 3, lines 13 - 15; lines 29-39).
>
> It would have been obvious to one of ordinary skill in the art at the time the invention was made to include these resource adapters in Li's shared memory space to ensure that memory usage gets saved by only running one resource adapter at a time and sharing it between applications.

Office Action dated August 20, 2008, p. 6.

Claim 5 is as follows:

> The method of claim 1, wherein the at least one shared web module includes at least one of a web archive (WAR) file, an enterprise java bean (EJB) archive file, and a resource archive (RAR) file.

Sharma teaches a resource adapter module, in the form of a resource archive file, may be shared among multiple J2EE applications as in:

> The stand alone deployment of a resource adapter module 206 into an application server 208 is typically done to support scenarios in which multiple J2EE applications share a single resource adapter module.

Sharma, col. 3 lines 13-17.

Therefore, *Sharma* provides a complete solution without need of the teaching of *Li* for sharing Java classes. One skilled in the art would therefore not be motivated to combine the teaching of *Sharma* with the teaching of *Li* and *Spotswood* as currently proposed. Further, *Sharma* teaches that the resource adapter archive is a collection packaged together as:

> A packaged resource adapter 202 includes Java classes and interfaces that are required for the implementation of both connector contracts and functionality of the resource adapter 202; utility Java classes for the resource adapter 202; native libraries required by the resource adapter 202; and any help files and documentation and descriptive meta information that ties all of the above elements together.

Sharma, col. 3 lines 29-35.

Sharma therefore teaches a package that includes more than the teaching of *Li* so that the teaching of *Li* would not be sufficient to support *Sharma*. Accordingly, *Li* does not add value to *Sharma*. Therefore, one skilled in the art would not combine the references as suggested.

The package of *Sharma* includes all of the elements needed for an adapter resource. In contrast, the claimed feature includes a reference to a shared module rather than the shared module itself. Therefore, *Sharma* teaches away from the claimed feature.

There is no motivation to combine *Sharma*. *Sharma* fails to teach the features missing from the combination of *Li* and *Spotswood*. *Sharma* teaches away from the claim feature. The combined teaching of *Li*, *Spotswood* and *Sharma* therefore fails to teach or suggest the claimed features of claim 5, including the features of claim 1 upon which claim 5 depends.

Therefore, the rejection of claim 5 under 35 U.S.C. § 103 has been overcome.

III. Conclusion

The subject application is patentable over the cited references. Therefore, the subject application should now be in condition for allowance. Applicants invite the Examiner to call the undersigned at the below-listed telephone number if, in the opinion of the Examiner, a telephone conference would expedite or aid the prosecution of this application.

DATE: November 19, 2008

Respectfully submitted,

/Cathrine K. Kinslow/

Cathrine K. Kinslow
Reg. No. 51,866
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants

CKK/wr